Convex Collider Creator

by Justin Couch / JustInvoke

Support email: justin@justinvoke.com
Support terms: https://justinvoke.com/contact/
Video Tutorial: https://youtu.be/Y8rQsWXzB2c

Table of Contents

Collider Editor Window

The editor window can be opened by navigating to *Window* > *Convex Collider Creator* in the toolbar or by pressing Alt+C (perhaps Option+C on Mac, untested). The use of this window is required in order to modify colliders. Certain window settings are automatically saved upon closing and loaded when opened.

Sections:

Editor Options

Reload Window On Compile – Whether to reload the window after script compilation. This is useful because code compilation can cause issues with the undo states associated with the window. The window will become undocked if it was previously docked.

GUI Width Scale – Scales the width of GUI elements in the window.

Mesh Preview Mode – Current method for previewing collision meshes.

> **None** – No mesh preview is drawn.
>
> **Real Mesh** – The actual generated mesh is drawn.
>
> **Box Approximation** – Gizmos are used to draw a box-shaped approximation of the collision mesh.

Live Preview – Whether to continuously update mesh previews.

Preview All Colliders – Whether to draw wireframe previews for all unselected meshes.

Live Edit Update Time – Time interval in seconds between preview updates.

Target Object

Target Is Selection – Whether the currently selected game object is the target for collider editing.

Null Selection Clears Target – Whether having no object selected will clear the target object in the editor window.

Use Collider Group – Whether to utilize a collider group component. The component is necessary to store collider properties for nondestructive editing and working with multiple colliders. If this is disabled, you can only modify one collider on the target object at a time and will not be able to save properties for future modifications (only the mesh collider component will remain). If this is disabled while a collider group is present on the target object, a dialog will appear to confirm deletion of the group component.

Target Object – The current game object for which colliders are being configured.

Collider index – Index of the collider being edited in a collider group. The max index indicates the index at which a new collider can be added to the group (at the end of the list of colliders). This collider does not yet exist in the group and undo states are not tracked as the properties are not yet serialized. In order to actually add the collider to the group, click the “Generate Collider” button below. Clicking the “Create New” button on the collider group component also works.

Is Trigger – Whether the collider is a trigger (the corresponding mesh collider component will be set as a trigger).

Material – The physic material to be placed on the corresponding mesh collider component.

Polygon Limit Test – The mode for testing the polygon count of the generated collision mesh. Generated collision meshes must not have more that 255 polygons. After a mesh is submitted to a convex mesh collider component, coplanar triangles are merged into larger polygons. This is why purely counting triangles is not sufficient.

> **Best Guess** – Most accurate estimate, small chance at not catching proper limit.
>
> **Safe Guess** – Less accurate but safer estimate.
>
> **Force Triangles** – Only counts triangles, safest but least accurate measure of polygons.
>
> **Force Quads** – Counts every pair of triangles as a quad, almost as safe as triangles but more accurate.
>
> **Force Edge Quads** – Estimates quads generated by corner and edge detail/roundness, least safe option.
>
> **Force Face Edge Quads** – Estimates quads generated by corner and edge detail/roundness as well as face subdivisions, more safe than just edge quads.

Bypass Polygon Test – Whether to ignore polygon count testing. Errors will be thrown if generated colliders have more than 255 polygons.

Auto Detail Reduction – Mode for automatically reducing the detail of meshes in order to fall under the polygon limit. This occurs during collider generation and is useful at runtime. Note that this is enabled by default and detail values will be reduced if they are too great when configuring new colliders.

> **None** – No detail reduction will occur.
>
> **All** – All detail values are reduced in succession. This helps to maintain proportions between detail values. The detail values chosen to be reduced cycle with each attempt in this order: face subdivisions, top/bottom segments, corner details. One of these 'groups' is reduced together at a time (e.g. all face subdivisions, top and bottom segments, all corners).
>
> **Largest First** – Largest detail values are reduced first. This generally requires fewer attempts. If multiple detail values are both equivalent and the greatest, they will be reduced together.

Detail Reduction Attempts – Maximum iterations for reducing detail before giving up and throwing a timeout error. Each attempt involves regenerating the collision mesh from scratch and testing the polygon count.

Buttons:

Generate Collider – Generates the currently selected collider and adds it to the target object with a mesh collider component.

Create Collider Group – Adds a collider group component to the selected game object. This button will replace the “Generate Collider” button if there is no collider group on the selected game object while “Use Collider Group” is true.

Duplicate Collider – Duplicates the currently selected collider, occupying the next index in the collider list.

Delete Collider – Deletes the currently selected collider.

Preset/Mesh Saving

Preset – Linked preset for saving and loading properties for the currently selected collider.

Mesh Path – Path in project for saving the collision mesh itself as an asset.

Buttons:

Save To Preset – Saves properties to the linked preset.

Load From Preset – Loads properties from the linked preset.

Save Mesh – Saves the collision mesh at the mesh path. The mesh must already be generated.

Save Mesh As – Opens a dialog to choose the mesh path and save to it.

Mirroring/Flipping

The flip buttons will flip a collider about the specified local axis relative to the target object.

The mirror buttons copy properties from one side of the collider and mirror them to replace the properties on the opposite side about the specified local axis. For example, clicking “Mirror +X” will copy everything on the positive x-axis (right) side of the collider to the negative x-axis (left) side while mirroring it. Clicking “Mirror -Y” will copy everything on the negative y-axis (bottom) side to the positive y-axis (top) side, mirrored.

Corners

Positions:

Position Edit Mode – The current mode for modifying the positions of corners.

> **Box Center** – Corners are positioned in a box shape defined by the center offset and size in local space.
>
> **Box Sides** – Corners are positioned in a box shape defined by positive and negative side extents. Each side of the box is extended or contracted freely.
>
> **Individual** – Corners are positioned freely in local space. Reverting to one of the box modes will overwrite individual position data. The move tool must be selected in order for position handles to be visible.

Each position mode has associated fields and scene view handles for modifying corner positions.

Radii:

Radius Handle Mode – Current mode for scene view handles to reflect either radii or radius offsets of each corner.

Radius Edit Mode – Current mode for corner radii editing.

> **Uniform** – All radii are set to the same value.
>
> **Uniform Individual** – Each corner radius is uniform and unique.
>
> **Advanced** – The radius of each corner about the x, y, and z-axes can be separately configured.

Max Radius – The maximum value allowed for radii editing.

Radius Offsets:

Radius offsets can be edited in a manner similar to radii. A corner's radius offset determines how its position is offset depending on its radius. For example, when the radius offset is set to 1, the corner will move itself as the radius is adjusted in order to keep the sides of the collider in the same place. When the radius offset is zero, the corner does not move as its radius is adjusted and the sides of the collider will expand or shrink accordingly. Values outside of the 0 to 1 range will lead to over-corrections.

Detail

Corner Detail Edit Mode – Current mode for editing lateral corner detail. Corners in this case refer to the front-right, front-left, back-right, and back-left corners. These are not the same as the corners described previously.

All – Each lateral corner detail is set to the same value.

Individual – Each lateral corner detail value is unique.

Top Segments – Lateral edge loops around the top side of the collider. If this is set to zero, it is recommended that the radii of the top corners be set to zero as well.

Bottom Segments – Lateral edge loops around the bottom side of the collider. If this is set to zero, it is recommended that the radii of the bottom corners be set to zero as well.

X-Y Plane Strips – Edge loops in the X-Y plane of the collider.

Y-Z Plane Strips – Edge loops in the Y-Z plane of the collider.

X-Z Plane Strips – Edge loops in the X-Z plane of the collider.

Detail Smoothness – Interpolation amount for smoothing vertices added by detail in between corners.

Upper/Lower Strip Distribution – These values adjust the vertical distribution of top/bottom segment edge loops. The default values are sufficient in most cases.

Deform Hooks

Deform hooks are used for fine-tuning the shape of colliders and are applied after all other properties have been used to calculate vertex positions. Hooks can be added and deleted with the “+” and “-” buttons in the bottom right of the hook list. The hook list is reorderable, meaning that each item in the list can be dragged up or down to change the order. Hook deformations are applied from top to bottom in the list, causing a hook to further deform vertices that have already been displaced by a previous hook.

Hook Properties:

Show Handles – Whether scene view handles for the hook are visible.

Type (dropdown menu) – The type of deformation performed by the hook.

Pull – Vertices are pulled in the direction the hook is pointing in local space.

Twist – Vertices are rotated about the center of the hook in local space to match the rotation of the hook.

Expand – Vertices are scaled away from or towards the center of the hook.

Enabled – Whether the hook is actively deforming vertices.

Name – Name for identifying the hook, optionally drawn as a label in the scene view.

Local Position – Position of the hook in the local space of the target object.

Local Rotation – Euler angle representation of the local rotation of the hook, stored internally as a quaternion.

Radius – The radius of effect around the hook in all directions. Vertices are deformed based on their distance to the hook.

Strength – The magnitude of vertex displacement. Negative values invert displacements. This is adjusted in the scene view with a magenta handle.

Falloff – Power applied to the deformation magnitude based on the distance of each vertex to the hook. This is adjusted in the scene view with a cyan handle.

Strength and falloff handles for hooks will scale the values as the handles are manipulated. This means that pulling/stretching a handle away will effectively multiply the current value of the strength or falloff. Negative strengths will grow negatively and positive strengths grow positively. Small values will change very little with large manipulations.

Reset Buttons

The reset buttons can be found at the very bottom of the editor window.

Reset Box/Corners – Corners are reset to the default box shape with default radii and radius offsets.

Reset Detail – Detail values are reset to their defaults.

Reset Hooks – All hooks are reset to their default values.

Delete All Hooks – All hooks are removed from the collider.

Delete All Colliders – All colliders on the target object will be deleted.

Collider Group Component

The collider group component stores collider information for editing and generation.

Generate Colliders on Start – Whether to generate all colliders in the group during the Start() event in the *MonoBehaviour*. This is useful when working with prefabs containing collider groups because generated collision meshes are not saved with prefabs (Unity does not serialize procedural meshes). This way you don't have to click “Generate Colliders” after dragging a prefab into the scene.

Preset Options

Preset – The current linked preset used for saving and loading collider group information.

Buttons:

Save To Preset – Saves the collider group's properties to the linked preset.

Load From Preset – Loads collider group properties from the linked preset.

Colliders

This sections lists all of the colliders in the group. The “Create New” button will add a new collider the group and open the editor window for modification. Each collider has a name field, edit button, duplicate button, and delete button.

The name field allows you to change the name of the collider in the group. The corresponding generated mesh will have this name. The edit button opens the editor window with the collider selected, the duplicate button duplicates the collider in the group, and the delete button removes the collider from the group.

Generate Colliders – This button generates all colliders in the group.

Draw Default Inspector

This section shows all serialized information on the component including collider properties. This is mainly for debug purposes and it's not recommended to modify values here directly.

Script Reference – Classes

Scripts in this asset are contained in the *ConvexColliderCreator* namespace. When referencing associated classes and functions, make sure to add “using ConvexColliderCreator;” to the top of your script(s).

Collider Generator

This class contains static methods for generating colliders.

Enumerations:

ColliderFinishStatus – Indicates either successful or failed collider generation. Some generation functions have parameters for passing an instance of this enum which is then modified to indicate the finish state of collider generation.

> **Success** – Generation succeeded.
>
> **Fail** – Generation failed for an unspecified reason.
>
> **FailTriCount** – Generation failed due to the collider having more than 255 polygons.
>
> **DetailTimeout** – Generation failed due to reaching the maximum number of detail reduction attempts. The collider still has more than 255 polygons.

PolygonTestMode – Modes used for testing polygon counts. See the “Polygon Limit Test” in the *Target Object* section of *Collider Editor Window* above. The enum matches the limit test modes, but with slightly different names.

PolygonTest – Actual modes used for testing polygon counts, same as *PolygonTestMode* but without *BestGuess* and *SafeGuess*. Internally, *BestGuess* chooses from either *EdgeQuads* or *FaceEdgeQuads*, and *SafeGuess* chooses between *TotalQuads* and *TotalTriangles*.

The final chosen mode depends on whether the collider is in a basic box shape. This requires that the corner position mode be set to either "Box Center" or "Box Sides," that both the radius and radius offset modes be set to "Uniform," and that the deform hooks list is empty. These settings are necessary for the sides of the box to be completely flat and thus each be singular polygons/quads.

Static Functions:

public static Mesh **GenerateCollider**(*ref* GeneratorProps **genProps**) – Generates a collision mesh with the indicated properties **genProps** and returns it. The generator properties must be initialized beforehand.

public static Mesh **GenerateCollider**(*ref* GeneratorProps **genProps**, *out* ColliderFinishStatus **cFin**) – Same as above, but **cFin** can be used for error handling, indicating either success or fail states following generation.

public static Mesh **GenerateCollider**(*ref* GeneratorProps **genProps**, *out* ColliderFinishStatus **cFin**, bool **preview**) – Same as above, but **preview** indicates whether this mesh is used for previewing collision meshes. This means that polygon count tests are bypassed.

Collider Group

See the *Collider Group Component* section above.

Public Variables:

generateOnStart – Whether all colliders on the group are generated during Start().

linkedPreset – Associated preset for saving and loading properties.

meshColComponents – List of mesh collider components on the game object. This is automatically populated.

colliders – List of collider instances in the group.

generating – True if asynchronous collider generation is in progress. Collider generation cannot be started if this is true.

Public Functions:

public ColliderInstance **AddCollider**() – Adds a new collider to the group with default properties.

public ColliderInstance **AddCollider**(GeneratorProps **gp**) – Adds a new collider to the group with the given properties **gp**.

public void **AddCollider**(ColliderInstance **ci**) – Adds an existing collider **ci** to the group.

public void **SetCollider**(int **index**, Mesh **m**, GeneratorProps **gp**) – Sets the collider at the **index** in the list to use the given mesh **m** and properties **gp**.

public void **DuplicateCollider**(int **index**) – Duplicates the collider at the **index** in the colliders list.

public void **DeleteCollider**(int **index**) – Deletes the collider at the **index** in the colliders list.

public void **DeleteCollider**(ColliderInstance **ci**) – Deletes a specified collider **ci** in the group if it exists.

public void **DeleteAllColliders**() – Deletes all colliders in the group.

public void **DestroyComponent**() – Clears all colliders and destroys the collider group component.

public void **ClearColliders**() – Removes all colliders from the group; does not remove mesh collider components.

public void **GenerateCollider**(int **index**) – Generates the collision mesh at the **index** in the collider list.

public void **GenerateAllColliders**() – Generates collision meshes for all colliders in the group

public void **GenerateAllColliders**(bool **async**) – Asynchronous collider generation method that uses a coroutine to stagger the generation of each collider. If **async** is true, each collider will be generated during a separate frame. If **async** is false, generation will be carried out normally as with the above function.

public void **SetColliderComponents**(bool **exitGUI**) – Sets up mesh collider components with proper collision meshes that have been generated. The **exitGUI** parameter is for preventing Unity from throwing errors with inspector drawing and is only necessary in the editor.

public bool **SaveToPreset**() – Saves collider group properties to the linked preset and returns true if successful.

public bool **LoadFromPreset**() – Loads collider group properties from the linked preset and returns true if successful.

Collider Instance

The collider instance class represents individual colliders in a collider group.

Public Variables:

name – The name of the collider and its generated mesh.

colMesh – The generated collision mesh.

props – Properties used for collider generation.

Public Functions:

public **ColliderInstance**() – Creates a new collider with default properties.

public **ColliderInstance**(string **n**) – Creates a new collider with the given name **n** and default properties.

public **ColliderInstance**(GeneratorProps **gp**) – Creates a new collider with the given properties **gp**.

public **ColliderInstance**(string **n**, GeneratorProps **gp**) – Creates a new collider with the given name **n** and properties **gp**.

public **ColliderInstance**(string **n**, Mesh **m**, GeneratorProps **gp**) – Creates a new collider with the given name **n**, mesh **m**, and properties **gp**.

public **ColliderInstance**(ColliderInstance **ci**) – Creates a new collider by copying an existing collider **ci**.

public void **GenerateCollider**() – Generates the collision mesh for this collider.

public bool **DestroyMesh**() – Destroys the mesh associated with this collider and returns true if successful.

Generator Props

This class contains properties that define the shape of individual colliders as well as functions for modifying certain properties. Public variables can be modified directly unless otherwise noted.

Public Variables:

linkedPreset – Associated preset for saving and loading properties.

meshAssetPath – Path in the project folder for saving a collision mesh to.

isTrigger – Whether the generated collider component is marked as a trigger.

physMat – The physic material applied to the generated collider component.

enum **CornerPositionEditMode** – Modes for modifying the positions of corners in the editor. Note that changing to one of the box modes with scripting will not automatically update corner positions. You must use either the SetBox() or SetCornerPosition() functions to update corner positions.

> **BoxCenter** – Positioned as box shape composed of box center position plus box size extents.

BoxSides – Positioned as box shape composed of side extents in each cardinal direction.

Individual – Each corner is positioned freely.

cornerPositionMode – The current mode for editing corner positions.

enum **CornerRadiusEditMode** – Modes for modifying the radii and radius offsets of of corners in the editor.

Uniform – Each corner has the same radius or radius offset.

UniformIndividual – Each corner has a unique radius or radius offset, but all axis components are the same.

Advanced – Each corner has a unique radius or radius offset and all axis components can be different.

cornerRadiusMode – The current mode for editing corner radii.

cornerOffsetMode – The current mode for editing corner radius offsets.

enum **CornerDetailEditMode** – Modes for modifying lateral corner detail values in the editor.

All – All detail values are the same.

Individual – All detail values are unique.

cornerDetailMode – The current mode for editing lateral corner detail values.

boxSize – Box extents for the *BoxCenter* positioning mode.

boxOffset – Box center position for the *BoxCenter* positioning mode.

boxSidesPos – Three box side extents in positive cardinal directions for the *BoxSides* mode.

boxSidesNeg – Three box side extents in negative cardinal directions for the *BoxSides* mode.

corners – Array containing the eight corners of the collider. This should not be modified.

enum **LateralCorner** { *FrontRight*, *FrontLeft*, *BackLeft*, *BackRight* } – Identifies the four lateral corners on which detail applies.

cornerDetails – Detail amounts for each lateral corner.

topSegments – Number of edge loops adding detail on the top of the collider.

bottomSegments – Number of edge loops adding detail on the bottom of the collider.

XYDetail – Number of detail edge loops in the X-Y plane.

YZDetail – Number of detail edge loops in the Y-Z plane.

XZDetail – Number of detail edge loops in the X-Z plane.

detailSmoothness – Smoothness of intermittent detail vertices between corners.

stripDistribution1 – Adjusts the distribution of vertex strips for the top and bottom segments. Higher values move strips closer to the top while lower values move strips closer to the bottom. This applies to the upper bound.

stripDistribution2 – Same as above but for the lower bound.

hooks – Array of hooks for deforming the collider.

bypassPolyTest – Whether to skip polygon count testing. This can lead to errors if detail values are too great.

polyTestMode – Current method of polygon testing. See the “Polygon Limit Test” in the *Target Object* section of *Collider Editor Window* above.

enum **DetailReductionMode** – Modes for automatically reducing the detail of the collider. See the “Auto Detail Reduction” in the *Target Object* section of *Collider Editor Window* above.

detailReduction – The current mode for reducing detail.

detailReductionAttempts – Maximum tries to reduce the detail before giving up and throwing a timeout error.

boxedCorners – Determines if the corners are aligned in a simple box shape. Do not set this directly. See “PolygonTest” in the *Collider Generator* section of *Script Reference – Classes* above.

Public Functions:

public **GeneratorProps**() – Creates a new instance of collider properties with defaults.

public **GeneratorProps**(Vector3 **size**, float **cornerRadius**) – Creates a new instance of collider properties in a box shape with the given **size** and radius **cornerRadius** for all corners.

public **GeneratorProps**(GeneratorProps **gp**) – Creates a new instance of collider properties by copying from an existing one **gp**.

public void **VerifyProperties**() – Makes sure properties are set correctly with valid values before generation takes place.

public void **CopyProperties**(GeneratorProps **gp**, bool **copyPreset**) – Copies properties from another *GeneratorProps* instance **gp**. If **copyPreset** is true, it will also copy the linked preset reference from the other instance.

public bool **SavePropertiesToPreset**() – Saves properties to the linked preset and returns true if successful.

public bool **LoadPropertiesFromPreset**() – Loads properties from the linked preset and returns true if successful.

public void **SetBox**(Vector3 **size**) – Sets the collider to be a box shape with the given **size**.

public void **SetBox**(Vector3 **offset**, Vector3 **size**) – Sets the collider to be a box shape with the given **size** and **offset** in local space.

public void **SetCornerPosition**(ColliderCorner.CornerId **corner**, Vector3 **pos**) – Sets the position of a corner with the given *CornerId* enum **corner** in local space to the position **pos**. See “enum CornerId” in the *Collider Corner* section of *Script Reference – Classes* below.

public void **SetCornerRadius**(ColliderCorner.CornerId **corner**, float **radius**) – Sets the radius of a corner with the given location identifier **corner** to the value of **radius**.

public void **SetCornerRadiiAll**(float **radius**) – Sets the radii of all corners to the value **radius**.

public void **SetCornerRadius**(ColliderCorner.CornerId **corner**, Vector3 **radius**) – Sets the components of the radius of a corner identified by **corner** to a Vector3 **radius**.

public void **SetCornerRadiiAll**(Vector3 **radius**) – Sets the components of the radii of all corners to a Vector3 **radius**.

public void **SetCornerRadiusOffset**(ColliderCorner.CornerId **corner**, float **offset**) – Sets the radius offset of a corner identified by **corner** to the value of **offset**.

public void **SetCornerRadiusOffsetsAll**(float **offset**) – Sets the radius offsets of all corners to the value of **offset**.

public void **SetCornerRadiusOffset**(ColliderCorner.CornerId **corner**, Vector3 **offset**) – Sets the components of the radius offset of a corner identified by **corner** to a Vector3 **offset**.

public void **SetCornerRadiusOffsetsAll**(Vector3 **offset**) – Sets the components of the radius offsets of all corners to a Vector3 **offset**.

public void **SetCornerDetail**(int **detail**) – Sets all lateral corner details to the value of **detail**.

public void **SetCornerDetail**(LateralCorner **corner**, int **detail**) – Sets the detail of a certain lateral corner identified by the given *LateralCorner* enum **corner** to the value of **detail**. See “enum LateralCorner” in the *Collider Instance* section of *Script Reference – Classes* above.

public void **SetCornerDetailsAll**(int **detail0**, int **detail1**, int **detail2**, int **detail3**) – Sets the details of all corners to the respective values **detail0**, **detail1**, **detail2**, and **detail3**. The index of each detail value matches the corner named by the respective index of the *LateralCorner* enum (e.g. if you were to cast the enum members to integers).

public Vector3 **GetAverageCornerPos**() – Returns the average local point between all corners.

public ColliderCorner **GetCornerAtLocation**(ColliderCorner.CornerId **cId**) – Returns the corner with the given location identifier **cId**.

public void **AddHook**(DeformHook **dh**) – Adds an existing deform hook **dh** to the collider's list of hooks.

public DeformHook **AddHook**(DeformHook.HookType **hType**, Vector3 **pos**, Quaternion **rot**, float **radius**, float **strength**, float **falloff**) – Adds a new hook with the given type **hType**, local position **pos**, local rotation **rot**, **radius**, **strength**, and **falloff** to the collider.

public DeformHook **AddHook**(string **name**, DeformHook.HookType **hType**, Vector3 **pos**, Quaternion **rot**, float **radius**, float **strength**, float **falloff**) – Same as above but the hook will take on **name** as its name.

public void **RemoveHook**(DeformHook **hook**) – Removes the specified deform **hook** from the collider if it exists.

public void **RemoveHook**(int **index**) – Removes a deform hook from the collider at the **index** in the hook list.

public void **ResetCorners**() – Resets the corners of the collider to form a basic box shape with default radii.

public void **ResetDetail**() – Resets all detail values to their defaults.

public void **SetCornerPositionsFromBox**() – Sets corner positions based on box properties.

public void **Mirror**(Vector3 **mirrorSide**) – Mirrors a collider by copying and mirroring the side of the collider indicated by the Vector3 **mirrorSide** to the opposite side. The input vector must be such that only one component is either 1 or -1 and the rest are 0.

public void **Mirror**(Vector3 **mirrorSide**, *ref* DeformHook[] **mHooks**) – Alternative mirror function with a supplied list of deform hooks **mHooks** to mirror. This is mainly for the editor window.

public void **Flip**(Vector3 **flipDir**) – Flips a collider in the specified direction **flipDir**. This does not copy properties from one side to the other, but swaps them instead. The input vector must be such that only one component is either 1 or -1 and the rest are 0.

public void **Flip**(Vector3 **flipDir**, *ref* DeformHook[] **mHooks**) – Alternative flip function with a supplied list of deform hooks **mHooks** to flip. This is mainly for the editor window.

Collider Corner

This class is for storing the properties of individual corners of colliders. Public functions and variables can be used to modify them unless otherwise noted. Collider corners should not be created manually for most use cases as the *GeneratorProps* class automatically creates its own corners.

Public Variables:

enum **CornerId** { *FrontTopRight*, *FrontTopLeft*, *FrontBottomRight*, *FrontBottomLeft*, *BackTopRight*, *BackTopLeft*, *BackBottomRight*, *BackBottomLeft* } – Location identifier for each collider corner.

cornerLocation – The current corner location. Do not assign directly.

normalizedCornerLocation – Normalized Vector3 representing the corner location. Do not assign directly.

localPos – Local position of the corner.

axleRadii – Component-wise radii of the corner.

radiusOffsets – Component-wise radius offsets of the corner.

Public Functions:

public **ColliderCorner**() – Creates a new collider corner with default properties.

public **ColliderCorner**(ColliderCorner **cc**) – Creates a new collider corner by copying an existing one **cc**.

public void **ResetRadii**() – Resets the corner radii and radius offsets to their default values.

public void **SetCornerLocationFromPosition**() – Sets the corner location based on the corner's local position; does not work if local position is zero on any axis.

public Vector3 **GetOffset**() – Returns the amount the center of the corner should be offset based on the radii and radius offsets.

public Vector3 **GetEdgePos**(Vector3 **dir**) – Returns the point on an edge of the corner in local space in the given direction **dir** based on the radii and radius offsets.

public void **SetRadiusProperty**(Vector3 **setProp**, bool **propToSet**, float **maxOffset**) – Sets either the radii or radius offsets of the corner to **setProp** and clamped by **maxOffset**. This will set the radii if **propToSet** is true, otherwise it will set the radius offsets if **propToSet** is false.

Deform Hook

Deform hooks are local-space objects that can be used to apply final transformations to generated collision meshes. The deformation amount is based on the distance between the hook and each vertex.

Public Variables:

name – Name of the hook for identification.

enabled – Whether the hook is actively deforming a mesh.

showHandles – Whether to show handles in the editor.

enum **HookType** – Possible deformation types.

> **Pull** – Vertices are pulled in the forward direction of the hook.
>
> **Twist** – Vertices are twisted around the hook based on its rotation. Default/identity rotation corresponds to no deformation.
>
> **Expand** – Vertices are scaled either away from or toward the hook based on the strength. A strength of 1 indicates no deformation, less than 1 shrinks, and greater than 1 expands. Rotation does not affect deformation.

hookType – The current deformation type.

localPos – Local position of the hook.

localRot – Local rotation of the hook.

localEulerRot – Euler representation of local rotation. If this is modified, SetRotationFromEuler() should be called to update the quaternion rotation.

radius – Radius around the hook in which vertices are deformed.

strength – Magnitude of deformation. Negative values invert deformation.

falloff – Falloff for deformation based on the distance between the hook and each vertex. This is basically a power applied to the distance.

set – Whether the hook properties have been set in the editor. This is mainly for use with the editor window.

Public Functions:

public **DeformHook**() – Creates a new deform hook with default values.

public **DeformHook**(string **n**, HookType **ht**, Vector3 **pos**, Quaternion **rot**, float **r**, float **s**, float **f**) – Creates a new deformation hook with the given name **n**, deformation type **ht**, local position **pos**, local rotation **rot**, radius **r**, strength **s**, and falloff **f**.

public **DeformHook**(DeformHook **h**) – Creates a new hook by copying an existing one **h**.

public void **Reset**() – Resets the hook's properties to their default values.

public void **Initialize**() – Sets up initial values, used with the editor window.

public void **SetRotationFromEuler**() – Sets the local quaternion rotation from the local Euler rotation.

public void **SetRotation**(Quaternion **rot**) – Sets the local rotation of the hook from the quaternion **rot** and updates the local Euler rotation to match it.

public void **SetRotation**(Vector3 **dir**) – Sets the local rotation of the hook from the Vector3 Euler rotation **dir** and updates the quaternion local rotation to match it.

Runtime Example

This script is used in the RuntimeExample scene and can be found on the game object named "Generator." The public functions in the script are used with the UI sliders and buttons in the scene (inside of the Canvas object). The scene and script serve as an example of procedural collider generation at runtime.

Public Variables:

target – The target object holding the generated collider. The script automatically sets up one generated collider on the object with one deform hook.

actionButtonText – Text object related to the drop/reset button. This reference is for the script to control how the text changes based on the state of the target object.

Public Functions:

public void **SetColliderLength**(float **length**) – Sets the z-axis size of the collider to **length**.

public void **SetColliderWidth**(float **width**) – Sets the x-axis size of the collider to **width**.

public void **SetColliderHeight**(float **height**) – Sets the y-axis size of the collider to **height**.

public void **SetColliderCornerRadii**(float **radius**) – Sets the radius of all corners on the collider to the value of **radius**.

public void **SetColliderDetail**(float **detail**) – Sets the detail properties of the collider to the value of **detail**.

public void **SetHookXPosition**(float **xPos**) – Sets the x-position of the deformation hook to **xPos**.

public void **SetHookYPosition**(float **yPos**) – Sets the y-position of the deformation hook to **yPos**.

public void **SetHookZPosition**(float **zPos**) – Sets the z-position of the deformation hook to **zPos**.

public void **SetHookRadius**(float **radius**) – Sets the radius of the deformation hook to **radius**.

public void **SetHookStrength**(float **strength**) – Sets the strength of the deformation hook to **strength**.

public void **RandomizeCollider**() – Randomizes the positions and radii of the corners.

public void **DoTargetAction**() – Either drops or resets the target object depending on its state.

Collider Preset

This class is for .asset files containing the serialized properties of a single collider. It inherits from the *ScriptableObject* class. For saving to and loading from collider presets, use the SavePropertiesToPreset() and LoadPropertiesFromPreset() functions in *GeneratorProps*. You can create a collider preset asset by navigating to the project window and choosing *Create > Convex Collider Creator > Collider Preset*.

props – *GeneratorProps* instance containing collider information.

Collider Group Preset

This class is for .asset files containing the serialized properties of multiple colliders. It inherits from the *ScriptableObject* class. For saving to and loading from collider group presets, use the SaveToPreset() and LoadFromPreset() functions in *ColliderGroup*. You can create a collider group preset asset by navigating to the project window and choosing *Create > Convex Collider Creator > Collider Group Preset*.

colliders – List of *ColliderInstance* instances, each element representing a single collider.

public void **ClearColliders**() – Clears all colliders in the preset and their generated meshes.